Wrapper classes

1. Check if character is a Digit

```
package Assessment_day7;

public class char_is_digit {

public static void main(String[] args) {
char c = '9' ;
try {
int digit = Integer.parseInt(String.valueOf(c));
System.out.println(c + " is a digit.");
} catch (NumberFormatException e) {
System.out.println(c + " is not a digit.");
}

}

}
```

Output:

9 is a digit.

2. Compare two Strings

```
package Assessment_day7;

public class Compare_two_string {
```

```
public static void main(String[] args) {
String str1 = "Welcome";
String str2 = "Welcome";
String str3 = "To";

System.out.println("Comparing str1 and str2= " +
str1.equals(str2));
System.out.println("Comparing str1 and str3= " +
str1.equals(str3));
}

}
```

Output:

```
Comparing str1 and str2= false
Comparing str1 and str3= false
```

3. Convert using valueof method

```
package Assessment_day7;

public class convert_string {

public static void main(String[] args) {
int num = 123;
double decimal = 45.67;
boolean flag = true;
char ch = 'A';

String strNum = Integer.valueOf(num).toString();
```

```
String strDecimal =
Double.valueOf(decimal).toString();
String strFlag = Boolean.valueOf(flag).toString();
String strChar = Character.valueOf(ch).toString();

System.out.println("Converted int to String: " +
strNum);
System.out.println("Converted double to String: " +
strDecimal);
System.out.println("Converted boolean to String: "
+ strFlag);
System.out.println("Converted char to String: " +
strChar);
}
}
```

Output:

```
Converted int to String: 123
Converted double to String: 45.67
Converted boolean to String: true
Converted char to String: A
```

4. Create Boolean Wrapper usage

```
package Assessment_day7;

public class Boolean_to_wrapper {

public static void main(String[] args) {
Boolean b=Boolean.TRUE;
System.out.println(b);
```

```
b=Boolean.valueOf("false");
System.out.println(b);

}

}
```

Output:

true
false

Pass by value and pass by reference

1. **Write a program where a method accepts an integer parameter and tries to change its value. Print the value before and after the method call.**

```
package Assessment_day7;

public class call_by_value {
public static void changeValue(int x) {
x= 5;
}
public static void main(String[] args) {
int num=6;
System.out.println("Before method call=" +num);
changeValue(num);
System.out.println("After method call=" + num);
}
```

```
}
```
Output:
Before method call=6
After method call=6

2. **Create a method that takes two integer values and swaps them. Show that the original values remain unchanged after the method call.**

```
package Assessment_day7;

public class Method_call {

public static void main(String[] args) {
int a = 10;
int b = 20;

System.out.println("Before swap:");
System.out.println("a = " + a);
System.out.println("b = " + b);

swap(a, b);

System.out.println("After swap:");
System.out.println("a = " + a);
System.out.println("b = " + b);
```

```
}

public static void swap(int a, int b) {
int temp = a;
a = b;
b = temp;
System.out.println("Inside swap method:");
System.out.println("a = " + a);
System.out.println("b = " + b);
}

}
```

Output:

```
Before swap:
a = 10
b = 20
Inside swap method:
a = 20
b = 10
After swap:
a = 10
b = 20
```

Call by Reference (Using Objects)

3. **Create a class Box with a variable length. Write a method that modifies the value of length by passing the Box object. Show that the original object is modified.**

```
package Assessment_day7;
class Box {
int length;
}

public class call_by_reference {
public static void change(Box b) {
b.length = 20;
}

public static void main(String[] args) {
Box b=new Box();
b.length=15;
System.out.println("Before=" +b.length);
change(b);
System.out.println("After=" +b.length);
}

}
```

Output:

Before=15

After=20

4. **Write a Java program to pass an object to a method and modify its internal fields. Verify that the changes reflect outside the method.**

```
package Assessment_day7;
class Employee {
String name;
int age;

public Employee(String name, int age) {
this.name = name;
this.age = age;
}

public void printDetails() {
System.out.println("Name: " + name);
System.out.println("Age: " + age);
}
}
public class Pass_object {

public static void main(String[] args) {
Employee emp = new Employee("Sanjana", 20);
System.out.println("Before modification");
emp.printDetails();

modifyEmployee(emp);
```

```
System.out.println("After modification");
emp.printDetails();
}

public static void modifyEmployee(Employee emp)
{
emp.name = "Sanjana";
emp.age = 25;
System.out.println("Inside modifyEmployee
method:");
emp.printDetails();
}
}
```

Output:

```
Before modification
Name: Sanjana
Age: 20
Inside modifyEmployee method
Name: Sanjana
Age: 25
After modification
Name: Sanjana
Age: 25
```

5. **Explain the difference between passing primitive and non-primitive types to methods in Java with examples.**

Primitive Types:
It is Passed by value Changes within a method do not affect the original value.
Example:-

```
package Assesement_day7;

public class primitive {
	public static void main(String[] args) {
		int num = 10;
		System.out.println("Before method call: " + num);
		modifyPrimitive(num);
		System.out.println("After method call: " + num);
	}
	public static void modifyPrimitive(int num) {
		num = 20;
		System.out.println("Inside method: " + num);
	}
}
```

Output:
Before method call: 10

Inside method: 20
After method call: 10

Non-Primitive Types

It is Passed by reference value. Changes within a method affect the original object.

```
class Person {
	String name;

	public Person(String name) {
		this.name = name;
	}
}

public class non_primitive {
	public static void main(String[] args) {
		Person person = new Person("Sanjana");
		System.out.println("Before method call="
+ person.name);
		modifyNonPrimitive(person);
		System.out.println("After method call= " +
person.name);
	}
```

```
	public static void modifyNonPrimitive(Person
person) {
		person.name = "Dhana";
		System.out.println("Inside method= " +
person.name);
	}
}
```

Output:

```
Before method call=Sanjana
Inside method=Dhana
After method call=Dhana
```

MultiThreading

1 Write a program to create a thread by extending the Thread class and print numbers from 1 to 5.

```
package Assessment_day7;
class MyThread extends Thread {
public void run() {
for (int i = 1; i <= 5; i++) {
System.out.println(i);
}
}
}
```

```
public class multithredding {

public static void main(String[] args) {
MyThread thread = new MyThread();
thread.start();

}

}
```

Output:

1
2
3
4
5

2 Create a thread by implementing the Runnable interface that prints the current thread name.

```
package Assessment_day7;
class MyRunnable implements Runnable {
public void run() {
System.out.println("Current Thread: " +
Thread.currentThread().getName());
}
}

public class runable {
```

```
public static void main(String[] args) {
Thread thread = new Thread(new MyRunnable());
thread.start();

}

}
```

Output:

Current Thread: Thread-0

3 Write a program to create two threads, each printing a different message 5 times.

```
package Assesement_day7;
class ThreadExample implements Runnable {
        private String message;
        public ThreadExample(String message) {
                this.message = message;
    }
            public void run() {
                for (int i = 0; i < 5; i++) {
                        System.out.println(message);
                        try {
```

```
                    Thread.sleep(100); // Pause for
100ms
                } catch (InterruptedException e) {

Thread.currentThread().interrupt();
                }
            }
        }
}


public class    Thread{
        public static void main(String[] args) {
            Thread thread1 = new Thread(new
ThreadExample("Hello from Thread 1"));
            Thread thread2 = new Thread(new
ThreadExample("Hello from Thread 2"));
            thread1.start();
            thread2.start();
        }
}
```

Output:

Hello from Thread 1

Hello from Thread 2

Hello from Thread 1

Hello from Thread 2

Hello from Thread 1

Hello from Thread 2

Hello from Thread 1

Hello from Thread 2

Hello from Thread 1

Hello from Thread 2

4 Demonstrate the use of Thread.sleep() by pausing execution between numbers from 1 to 3.

```
package Assessment_day7;

public class multi {
public static void main(String[] args) {
try {
for (int i = 1; i <= 3; i++) {
System.out.println(i);
Thread.sleep(1000);
}
} catch (InterruptedException e) {
Thread.currentThread().interrupt();
```

```
}
}
}
```

Output:

1

2

3

5 Show how to stop a thread using a boolean flag.

```
package Assesement_day7;
class MyThread extends Thread {
	private boolean running = true;
	public void stopThread() {
		running = false;
	}
	public void run() {
		int i = 0;
		while (running) {
			System.out.println("Thread is
running: " + i);
			i++;
			try {
```

```
                    Thread.sleep(100); // Pause for
100ms
                } catch (InterruptedException e) {

Thread.currentThread().interrupt();
                }
                if (i >= 10) {
                    Break;
                }
            }
            System.out.println("Thread stopped.");
        }
}

public class stop_thread {
        public static void main(String[] args) throws
InterruptedException {
            MyThread thread = new MyThread();
            thread.start();

            Thread.sleep(500);
```

```
            thread.stopThread();
        }
}
```

Output:

```
Thread is running: 0
Thread is running: 1
Thread is running: 2
Thread is running: 3
Thread is running: 4
Thread stopped.
```

6 Create a program with multiple threads that access a shared counter without synchronization. Show the race condition.

```
package Assesement_day7;
class Counter {
        private int count = 0;
        public void increment() {
            count++;
        }
        public int getCount() {
            return count;
```

```
    }
}
class CounterThread extends Thread {
    private Counter counter;

    public CounterThread(Counter counter) {
        this.counter = counter;
    }
    public void run() {
        for (int i = 0; i < 10000; i++) {
            counter.increment();
        }
    }
}

public class Thread {
    public static void main(String[] args) throws
InterruptedException {
        Counter counter = new Counter();
        CounterThread thread1 = new
CounterThread(counter);
```

```
        CounterThread thread2 = new
CounterThread(counter);

        thread1.start();
        thread2.start();

        thread1.join();
        thread2.join();

        System.out.println("Expected count:
20000");
        System.out.println("Actual count: " +
counter.getCount());
    }
}
```

Output:

Expected count: 20000

Actual count: less than 20000

7 Solve the above problem using synchronized keyword to prevent race condition.

```
package Assesement_day7;
class Counter {
	private int count = 0;
	public void increment() {
		synchronized (this) {
			count++;
		}
	}

	public synchronized int getCount() {
		return count;
	}
}

class CounterThread extends Thread {
	private Counter counter;
	public CounterThread(Counter counter) {
		this.counter = counter;
	}
	public void run() {
		for (int i = 0; i < 10000; i++) {
```

```
                counter.increment();
            }
        }
}
public class Thread {
    public static void main(String[] args) throws
InterruptedException {
            Counter counter = new Counter();
            CounterThread thread1 = new
CounterThread(counter);
            CounterThread thread2 = new
CounterThread(counter);

            thread1.start();
            thread2.start();

            thread1.join();
            thread2.join();

            System.out.println("Expected count:
20000");
```

```
            System.out.println("Actual count: " +
counter.getCount());
        }
}
```

Output:

```
Expected count: 20000
Actual count: 20000
```